Санкт–Петербургский государственный университет
Кафедра компьютерного моделирования и многопроцессорных систем

Малеванный Даниил Михайлович

Выпускная квалификационная работа бакалавра

Разработка технологии моделирования взаимодействия распределенных реестров и приложений с использованием платформы контейнеризации

Направление 01.03.02
«Прикладная математика и информатика»

Научный руководитель:
кандидат технических наук,
Гришкин В.М.

Санкт-Петербург
2019 г.

Содержание

Введение 3

Постановка задачи 5

Обзор литературы 6

Глава 1. Система моделирования 7

1.1. Моделирование работы узлов приложения 7

1.2. Моделироваие сетевых взаимодействий 8

1.3. Распределение нагрузки на узлы кластера 10

Глава 2. Программный интерфейс описания и запуска модели 11

2.1. Общие принципы организации API 11

2.2. Описание узлов приложения и логической структуры сети 11

2.3. Настройка динамической маршрутизации 12

2.4. Создание виртуальных локальных сетей 13

2.5. Запуск и остановка модели 13

Глава 3. Пользовательский интерфейс мониторинга и управления моделью 14

3.1. Обзор пользовательского интерфейса системы моделирования 14

3.2. Управление качеством работы виртуальной сети 15

3.3. Программный интерфейс подсистемы мониторинга 16

3.4. Пользовательский интерфейс подсистемы мониторинга . . 18

Глава 4. Демонстрация работы системы 20

4.1. Подготовка приложения для использования в системе . . 20

4.2. Описание моделей, используемых в тестировании 21

4.3. Ход тестирования приложения 22

4.4. Результаты тестирования 23

Выводы 25

Заключение 28

Список литературы 28

Введение

Закон Мура гласит, что количество транзисторов на чипе микропроцессора удваивается каждые два года в ходе совершенствования технологии производства [1]. Впервые сформулированный в 1965 году, он полвека диктовал темп развития вычислительной техники. Однако проблемы, связанные с выводом генерируемого транзисторами тепла, не позволяют производителям продолжать удваивать их количество на чипе [1]. Связанное с этим замедление темпа увеличения производительности вычислительной техники заставляет разработчиков искать новые пути развития для повышения производительности их продуктов [2]. Одним из способов, способных обеспечить необходимый эффект, является распределение вычислительной нагрузки на несколько компьютеров. Основанные на этом принципе системы называются распределёнными, а используемые для непосредственных вычислений машины — узлами системы. По мере увеличения пропускной способности глобальной сети распределённые системы получают всё большее распространение [3][4].

Вне зависимости от используемого в распределённой системе алгоритма организации совместной работы и разбиения нагрузки, её узлы будут использовать сетевые соединения для координации рабочего процесса. При разработке такой системы необходимо обеспечить её тестирование с учётом логической структуры и различных возможных состояний физической сети, поверх которой должно будет работать приложение. (Под логической структурой сети здесь имеется в виду способ объединения узлов в группы, используемый для организации её работы) Проведение такого тестирования связано с большими трудностями, однако, если проигнорировать его и предположить все узлы напрямую соединёнными между собой идеальной сетью, то даже успешное тестирование приложения сможет гарантировать его правильную работу в реальных условиях только при нормальном состоянии сети и не позволяет оценить влияние на него проблем сетевого плана.

Так, для децентрализованных сервисов, построенных на платформе Ethereum, было показано, что изменение скорости передачи сообщений

между узлами может повлиять на соотношение их мощностей при майнинге [5]. Для другой распределённой системы, предполагавшейся защищённой по свой архитектуре криптовалюты “Bitcoin” [6], существуют доказательства проведения успешных атак, основанных на манипуляциях с протоколами динамической маршрутизации пакетов в глобальной сети, поверх которой работает система [7].

Данная работа описывает основанную на технологии контейнеризации систему симуляции распределённых систем MADT ("Modeling and Analysys of Distributed Technologies"), обеспечивающую возможность тестирования их работы при изменяющемся поведении сети и анализа влияния логической структуры сети на такие системы. Система моделирует работу узлов системы и логическую структуру нижележащей сети и позволяет динамически манипулировать качеством её работы. В качестве вычислительных ресурсов для моделирования система может использовать как одну машину, так и кластер, таким образом позволяя моделировать работу систем со значительным (более 1000) количеством узлов. Для сравнения, количество полноценных узлов во всей сети Bitcoin оценивается как 10000 [8]. Для доступа к информации, получаемой в ходе тестирования, была реализована система мониторинга, визуализирующая данные, генерируемые моделью, в реальном времени и с учётом структуры сети, позволяя соотносить изменения в работе сети с изменениями в поведении системы и упрощая планирование дальнейшего направления тестирования.

В качестве демонстрации работы созданной системы симуляции был проведён анализ поведения узлов DHT Kademlia при нарушении работы сети, в ходе которого были обнаружены временное замедление работы всей сети при отключении одного узла, продолжение работы отсечённого от остальной сети набора узлов и невозможность переподключения узла после кратковременного нарушения его соединения с остальной сетью, что вместе позволяет владельцу физической сети разделять глобальную сеть DHT Kademlia на локальные участки посредством кратковременных нарушений передачи сообщений.

Постановка задачи

Целью данной работы является реализация системы симуляции работы узлов распределённой системы и сетевых взаимодействий между ними, предназначенной в первую очередь для тестирования устойчивости таких систем относительно атак сетевого плана. Для избежания возможного замешательства, моделируемая распределённая система далее будет упоминаться как “приложение”. При этом для необходимого уровня реалистичности симуляции рабочего процесса распределённого приложения может понадобится одновременное моделирование большого количества его узлов. Система должна учитывать это и минимизировать накладные вычислительные расходы, связанные с симуляцией работы узлов моделируемых приложений.

Виртуальные сети, соединяющие отдельные узлы системы, не должны требовать от приложения дополнительной настройки, а в своём поведении и логической структуре должны быть близки к реальным сетям. Для этого система симуляции должна предоставлять возможность создания локальных сетей, обеспечивать узлам, находящимся в них доступ к глобальной сети посредством маскарадинга IP-адресов и моделировать работу протоколов динамической маршрутизации, обеспечивающих работу реальной глобальной сети [9]. Для изучения влияния логической структуры нижележащей сети на работу приложения целесообразно предоставление пользователям инструментария для описания логической структуры виртуальных сетей, который позволил бы им генерировать модели, основанные на сколь угодно сложных сетях.

Кроме наблюдения работы в стабильных условиях, для проведения полноценного тестирования распределённого приложения необходимо изучение влияния на него изменений качества работы нижележащей сети. При этом нужно учитывать, что такие изменения могут повлечь за собой изменения в работе протоколов динамической маршрутизации. Таким образом, система должна предоставлять пользователям возможность динамически изменять стабильность и скорость передачи сообщений в отдельных участках виртуальной сети, так как изменение качества работы всей сети не

повлияет на динамическую маршрутизацию трафика в ней. Для анализа пользователем влияния вносимых изменений на моделируемое приложение необходима подсистема мониторинга, визуально отображающая состояние всех узлов моделируемого приложения в реальном времени. Так как моделирование приложений большого масштаба может потребовать создания виртуальных сетей высокой сложности с большим количеством узлов, подсистема мониторинга должна визуализировать состояние приложения с учётом структуры сети и быть максимально масштабируемой.

Для упрощения работы пользователей с системой и предоставления им возможности сравнивать результаты тестирования различных приложений в специфичных условиях, система должна требовать минимальных трудозатрат на адаптацию распределённых приложений для работы в модели.

Наконец, для демонстрации возможностей созданной системы моделирования необходим пример её работы, демонстрирующий описание логической структуры виртуальной сети и тестирование широко распространённого и лёгкого для понимания распределённого приложения, включая его адаптацию для использовании в системе, интеграцию с описанной ранее сетью, запуск и наблюдение работы модели.

Обзор литературы

Существует большое количество специализированных симуляторов распределённых приложений, таких как NS-3 [10] и PeerSim [11], предоставляющих пользователям широкие возможности для моделирования распределённых систем, но требующих от пользователей тщательного исследования документации и сложной адаптации тестируемых систем для использования в модели, что значительно усложняет сравнительное тестирование работы нескольких технологий.

Другие системы симуляции распределённых систем, такие как PeerfactSim.K[12] и BLOCKBENCH [13] ориентированы на бенчмаркинг решений и не позволяют динамически изменять параметры модели, и, следовательно, не подходят для тестирования устойчивости распределённых систем.

Ближайшим по методам и целям является проект VIBES [14], однако он направлен исключительно на симуляцию работы систем, основанных на технологии блокчейн и не может использоваться для изучения работы других распределённых систем, например DHT или менеджеров вычислительных кластеров.

Глава 1. Система моделирования

1.1 Моделирование работы узлов приложения

Как уже было сказано ранее, каждый узел распределённого приложения — это изолированная машина. Стандартным подходом при необходимости моделирования работы отдельного компьютера является использование виртуальных машин. Однако в целях минимизации вычислительных ресурсов, необходимых для моделирования каждого узла приложения, при сохранении их изоляции друг от друга MADT использует технологию контейнеризации. В качестве конкретной платформы был выбран Docker [15], так как он является де-факто стандартным решением индустрии, для него существует библиотека образов, используемых при создании контейнеров, и он предоставляет необходимый для организации сетевых взаимодействий контейнеров функционал, который при использовании других платформ контейнеризации необходимо реализовывать самостоятельно.

Docker реализован как сервер, работающий в системе на заднем плане и принимающий запросы на разных протоколах, и несколько видов клиентов, которые реализуют пользовательский интерфейс и управляют действиями сервера по одному из поддерживаемых протоколов. MADT использует в качестве клиента docker программный интерфейс на языке python [16], который может подключаться к серверу как через UNIX-сокет, так и через TCP-порт. Это позволяет MADT разворачивать контейнеры модели на распределённом кластере, управляя docker-серверами через TCP-подключения.

Подготовка приложения к тестированию в системе, таким образом, заключается в создании Docker-образа и сценария тестирования, который

должен описывать бесконечный цикл, на каждой итерации которого проверяется интересующий аспект работы приложения. Первая часть этого процесса хорошо описана как непосредственно в документации Docker, так и в большом количестве руководств на сторонних ресурсах. Вторая часть, в свою очередь, может быть осуществлена на основе функциональных тестов, если они уже имеются у тестируемого приложения.

1.2 Моделироваие сетевых взаимодействий

Как уже упоминалось ранее, под логическим разделением сети понимается способ объединения узлов в группы, используемый для организации её работы. Данная работа главным образом сконцентрирована на моделировании сетей, работающих по протоколам TCP/IP. В данном случае логической структурой сети является её разделение на подсети, которым также, как и узлам, выданы IP-адреса.

Моделирование сети между узлами приложения основано на этой идее. Система достоверно моделирует сетевые взаимодействия вплоть до сетевого уровня, которому принадлежат роутеры и шлюзы, моделирование функционала которых представляет особый интерес в данном проекте. Моделирование взаимодействий и устройств более низкого канального уровня представляет меньший интерес и связано с увеличением вычислительной мощности, необходимой для работы модели.

Docker позволяет соединить контейнеры на сетевом уровне, создавая виртуальные IP-подсети. Однако, он не позволяет указывать IP-адрес и маску виртуальной подсети и выдаёт их самостоятельно из внутреннего пула, который содержит IP-адреса из диапазона 172.17.0.0 - 172.17.255.255.

Для устранения зависимости системы моделирования от состояния пула адресов Docker необходимо обеспечить дополнительную настройку созданных при подключении к docker-сетям виртуальных интерфейсов контейнеров. Для этого через интерфейс, предоставляемый docker, внутри контейнера запускаются команды системной утилиты ip (8), которые назначают сетевым интерфейсам дополнительные ip-адреса, заранее определённые системой моделирования.

В случае, если модель приложения разворачивается на нескольких машинах, использование сетевого функционала docker становится невозможным, так как он основан на работе с ядром операционной системы хоста. Для обеспечения работы виртуальных сетей между контейнерами, расположенными на разных узлах кластера, был использован WireGuard VPN [17]. Как и у других реализаций VPN его функция - объединение компьютеров, находящихся в разных сегментах сети в виртуальную частную сеть (Virtual Private Network) посредством туннелирования соединений. Главным преимуществом WireGuard является его реализация в виде модуля ядра Linux. Это позволяет создавать виртуальные интерфейсы WireGuard внутри сетевого пространства имен контейнера, даже если WireGuard недоступен в нём. Таким образом, для подключения контейнера к VPN нет необходимости изменять его docker-образ, что облегчает подготовку приложения для тестирования в MADT.

Моделирование динамического роутинга

Для обеспечения динамической маршрутизации трафика между виртуальными подсетями модели используется сюита Quagga [18], предоставляющая набор реализаций различных протоколов динамической маршрутизации, включая RIP, OSPF и BGP. Конфигурация Quagga для протоколов RIP и OSPF генерируется системой по заранее известным IP-адресам подсетей и узлов сети, которые в свою очередь зависят от конфигурации протокола BGP.

Роутеры, соединяющие подсети, представляют из себя контейнеры с демонами Quagga, запущенными внутри, и подключенные к подсетям аналогично остальным контейнерам. Docker-образ для контейнеров роутеров поставляется как часть системы MADT, так что их использование не требует от пользователей никаких дополнительных действий.

Моделирование IP-маскарадинга

Межсетевой экран OS Linux обладает встроенными средствами для организации IP-маскарадинга между локальной и глобальной сетью. Для настройки NAT система использует заранее известные IP-адреса подсетей, при этом соответствующие команды утилиты iptables, использующейся для управления межсетевым экраном, выполняются в контейнере вместе с на-

стройкой альтернативных IP, как было указано ранее.

Настройка системной маршрутизации

Использование динамической маршрутизации трафика и IP-маскарадинга на шлюзах локальных сетей требует дополнительной настройки системной маршрутизации внутри docker-контейнеров, соответствующих узлам моделируемого приложения. Она осуществляется с помощью стандартной утилиты ip (8) вместе с настройкой альтернативных IP-адресов и NAT. Это позволяет уменьшить количество запросов к серверу docker и ускорить настройку модели. Наличие утилит iptables и ip — единственное требование к docker-образу для использования в системе.

1.3 Распределение нагрузки на узлы кластера

Для организации работы модели на вычислительном кластере необходимо определить алгоритм распределения docker-контейнеров модели по узлам кластера. Целью работы алгоритма является создание такого распределения контейнеров по узлам, при котором одновременно будет минимизировано количество виртуальных подсетей, соединяющих работающие на разных узлах кластера контейнеры, и будет достигнута максимальная равномерность распределения контейнеров. Это позволит снизить количество виртуальных подсетей с завышенной из-за использования VPN задержкой передачи сообщений и максимально эффективно использовать вычислительные ресурсы кластера.

Алгоритм, использованный в системе основан на том факте, что сеть по своей структуре является гиперграфом: узлы сети соответствуют вершинам гиперграфа, а виртуальные подсети - его рёбрам, соединяющим наборы вершин. Сформулированная ранее проблема в теории гиперграфов известна как задача создания сбалансированного разреза гиперграфа на N частей. Доказано, что она является NP-полной [19]. Запуск на кластере предназначен для сложных моделей, состоящих из большого количества узлов и подсетей. Так как поиск достоверно оптимального решения для них может занимать неприемлемо большое время, был использован эвристический алгоритм HYPE [19], позволяющий ускорить построение распре-

деления контейнеров по узлам кластера.

Глава 2. Программный интерфейс описания и запуска модели

2.1 Общие принципы организации API

В целях ускорения запуска сложных моделей с большим количеством узлов и предоставления пользователям возможности строить модели не имея доступа к вычислительным мощностям, необходимым для их запуска, интерфейс, выполненный в виде библиотеки на языке Python, разделён на две независимые части, одна из которых предназначена только для описания модели, а вторая - только для запуска и остановки. Сама модель передаётся между ними в собранном виде, который содержит только выделенные IP-адреса узлов и подсетей, распределение узлов по подсетям, конфигурации для Quagga и параметры запуска контейнеров и системной маршрутизации.

Так как логика работы системы запуска модели детально описана в прошлой главе, эта будет больше сконцентрирована на программном интерфейсе описания модели.

2.2 Описание узлов приложения и логической структуры сети

Узлы модели

Как было уже сказано ранее, каждому узлу модели соответствует docker-контейнер, запускаемый через docker-клиент, реализованный на языке Python. Построенный программный интерфейс создания узла приложения, принимает те же опции, что и интерфейс создания контейнера клиента docker, для последующей передачи в него. Единственным отклонением является возможность заранее определить, должен ли контейнер иметь доступ к миру вне симуляции, так как эта возможность не всегда нужна, но значительно усложняет правила системной маршрутизации внутри контейнера и, таким образом, замедляет запуск системы.

Стоит также отметить, что роутерам, соединяющим сегменты виртуальной сети между собой, так же соответствуют docker-контейнеры. Отдельный интерфейс для их создания полностью повторял бы описанный выше. В связи с этим интерфейсы создания узлов приложения и виртуальных роутеров объединены в один.

Виртуальные подсети

В отличии от узлов приложения, виртуальным подсетям не всегда соответствует виртуальная сеть docker, поэтому логика работы интерфейса для их создания отличается от описанного выше. Вместо этого он позволяет пользователю только перечислить узлы, входящие в сеть и шлюз по умолчанию для них в случае необходимости. Если он не указан, система выберет его сама из роутеров, присутствующих в подсети.

2.3 Настройка динамической маршрутизации

Для настройки работы протоколов динамической маршрутизации в интерфейсе была введена абстрация оверлея, под которым понимается набор роутеров из соседних сегментов сети, которые непосредственно обеспечивают работу протокола. С точки зрения пользователя, организация работы протоколов RIP и OSPF заключается только в создании такого оверлея путем перечисления роутеров. Однако для работы протокола BGP (Border Gateway Protocol) необходима дополнительная информация помимо перечня роутеров.

Так как логика работы протокола основана на заранее известном разделении сети на автономные системы, каждой из которых соответствует свой пул IP-адресов, при создании оверлея для BGP необходимо перечислить не только роутеры, но и входящие в соответствующие им автономные системы узлы сети. При последующей сборке модели, данные о разделении сети на автономные системы будет использовано при распределении IP-адресов узлов и подсетей.

2.4 Создание виртуальных локальных сетей

Несмотря на то, что при запуске модели связь локальной сети и глобальной виртуальной сети обеспечивается одной командой, с точки зрения описания модели создание локальных сетей выглядит гораздо сложнее. Локальная сеть должна реализовывать полный функционал глобальной: создание узлов, подсетей, оверлеев динамической маршрутизации и локальных сетей. Это достигнуто путём определения класса сети, который реализует весь описанный ранее функционал и контролирует наличие только одной глобальной сети. Главным отличием локальных сетей от глобальной заключается в наличии шлюза, который соединён с глобальной сетью и перенаправляет в неё трафик из локальной сети, при этом осуществляя маскарадинг IP-адресов. Для работы такого шлюза весь трафик из подсетей локальной сети, сопряжённых с ним, перенаправляется в него. Сборка модели происходит рекурсивно, начиная от глобальной сети и переходя к локальным.

2.5 Запуск и остановка модели

Программный интерфейс, осуществляющий запуск и остановку модели, состоит из трёх функций, которые осуществляют соответственно старт, остановку и перезапуск работы модели. Непосредственная логика работы функций зависит от условий запуска модели и определена в отдельных модулях для работы на одном узле и на кластере. При этом добавление новых модулей не требует изменение интерфейса. Такая структура улучшает читаемость исходного кода системы и упрощает её дальнейшее усовершенствование.

Глава 3. Пользовательский интерфейс мониторинга и управления моделью

3.1 Обзор пользовательского интерфейса системы моделирования

Пользовательский интерфейс системы MADT реализован в виде веб-приложения с помощью микрофреймворка Flask [20] на языке Python. Он предоставляет листинг доступных для запуска моделей, изображённый на рис 1, листинг работающих в рамках отдельной модели контейнеров, изображенный на рис **??**, предоставление пользовательского интерфейса для API запуска и остановки моделей, описанного ранее, интерфейса управления качеством работы виртуальной сети и панели мониторинга работы моделируемого приложения. Данные о доступных моделях интерфейс получает из листинга заранее определённой директории, о работающих контейнерах - через клиент docker. Создание пользовательского интерфейса для запуска и остановки модели также является тривиальной задачей.

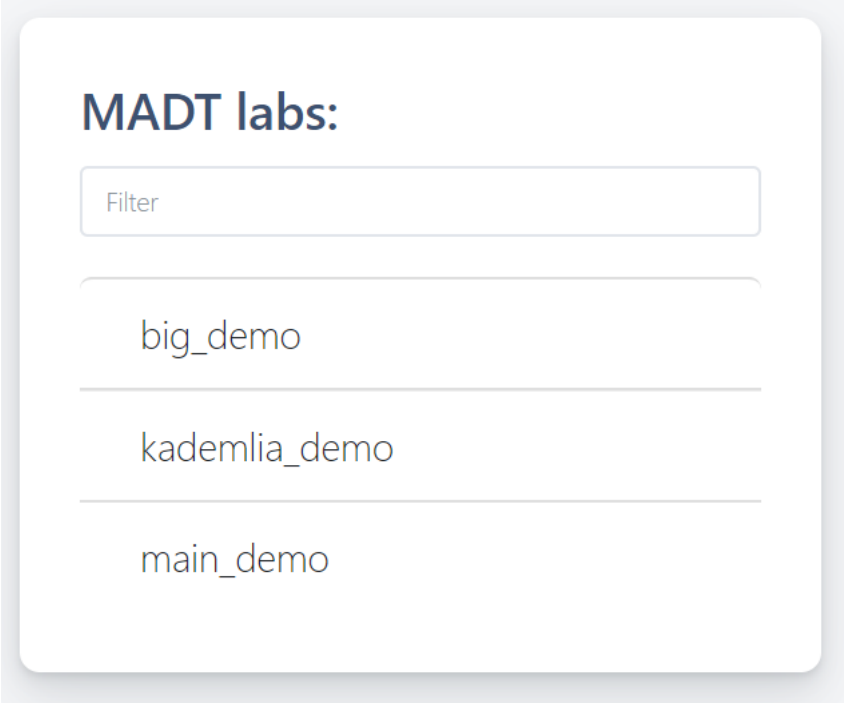

Рис. 1: Домашняя страница, показывающие модели, доступные для запуска

Стоит также отметить дополнительную функцию веб-сервиса, не свя-

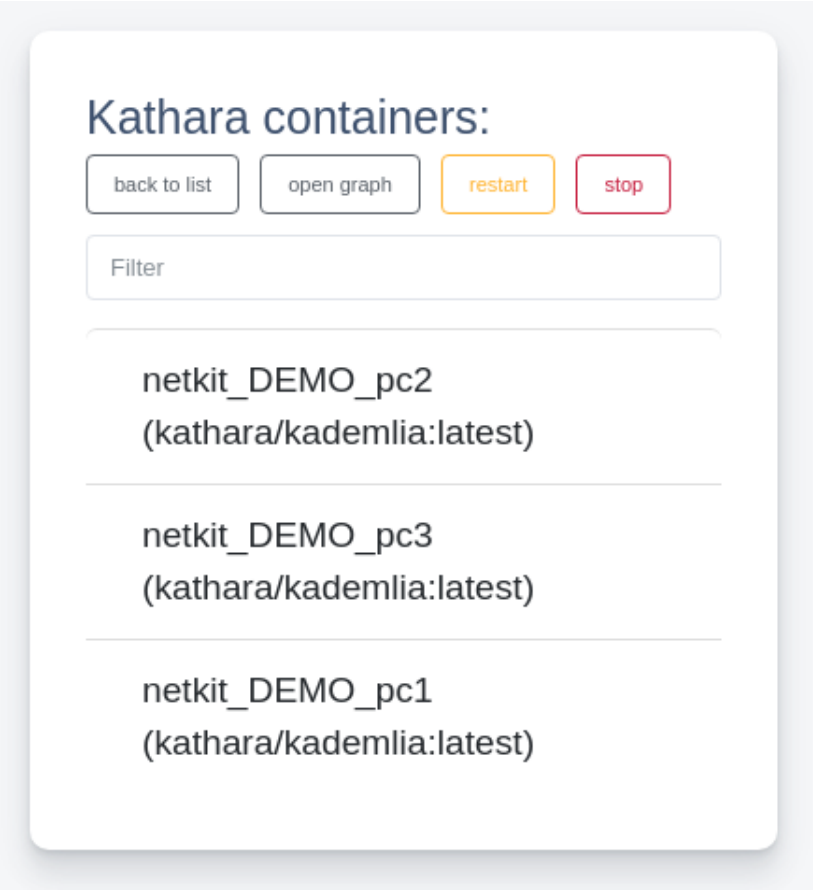

Рис. 2: Страница модели, отображающая работающие в её рамках контейнеры и их docker-образы

занную с пользовательским интерфейсом — в случае, когда для моделирования используется рапределённый кластер, он принимает от его узлов запросы на добавление в систему и сохраняет их параметры. Кроме того, он хранит полученное при запуске модели распределение контейнеров и сетей по узлам кластера, которое затем будет необходимо для её остановки.

Продолжение данной главы будет сконцентрировано на описании двух наиболее содержательных функций пользовательского интерфейса - настройке качества работы виртуальной сети и отображении состояния моделируемого приложения в реальном времени.

3.2 Управление качеством работы виртуальной сети

Для предоставления пользователям системы возможности изучения воздействия качества работы виртуальный сети на тестируемое приложе-

ние был реализован веб-интерфейс управления качеством работы сети. В качестве основы для него был использован пакет tcconfig [21], изменяющий качество работы сетевых интерфейсов манипулируя очередями сообщений ядра системы Linux через утилиту tc (8). Tcconfig позволяет изменять пропускную способность интерфейса, задержку передачи сообщений, процент потери, повреждения, повторения и перестановки пакетов, проходящих через него. Значительным преимуществом этого пакета является поддержка настройки работы не только сетевых интерфейсов, но и контейнеров docker. При этом нет никаких требований к программному обеспечению, доступному в контейнере, а накладываемый эффект оказывает влияние на все его сетевые интерфейсы.

Для предоставления удобного пользовательского интерфейса к консольным командам из этого пакета была разработана веб-форма, которая при загрузке отображает актуальные параметры наложенных ограничений и позволяет выставлять новые. Интерфейс веб-формы показан на рис. 3

3.3 Программный интерфейс подсистемы мониторинга

Для передачи узлами моделируемой распределённой системы за пределы симуляции своего состояния во время тестирования был разработан программный интерфейс, основанный на библиотеке ZMQ, предназначенной для организации обмена сообщениями. Передача сообщений происходит через IPC-сокет, расположенный во всех контейнерах по пути /lab/lab.sock. При этом папка /lab смонтирована в контейнер из заранее заданного расположения на хосте, также общего для всех контейнеров на одном хосте. Таким образом, физический сокет будет единым для всех узлов моделируемого приложения, расположенном на одном хосте. В случае разворачивания модели на кластере сокет туннелируется из центральной машины системы на все узлы кластера с помощью системной утилиты socat. Наличие единственной точки доступа для передачи сообщений улучшает масштабируемость системы и уменьшает задержки при чтении сообщений.

Сообщения, передаваемые узлами приложения, должны иметь фор-

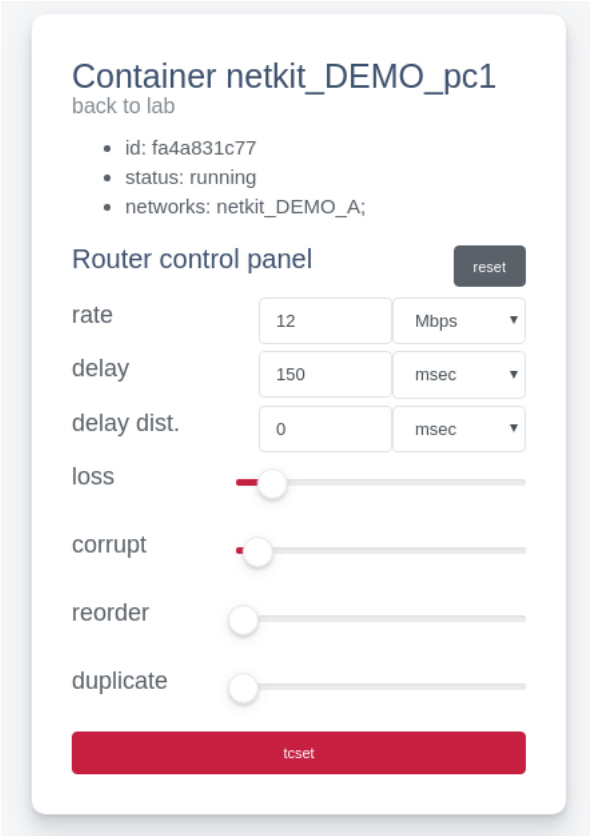

Рис. 3: Страница контейнера, отображающая текущие ограничения его сетевых соединений и позволяющая накладывать новые

мат JSON и содержать объект, состоящий из трёх полей:

- status — статус операции, допустимые значения — целое число от 0 до 3
- traffic — количество обработанных данных, допустимо любое положительное целое число
- log — вывод при выполнении операции, допустима любая строка
- hostname — имя узла, должно быть равно переменной окружения $HOSTNAME и используется для идентификации источника сообщения

Предполагается, что компоненты тестируемого приложения будут отправлять сообщения после каждой завершённой итерации тестирования

работы всей системы. В качестве примера такой итерации можно привести попытку клиента Kademlia сохранить и затем загрузить файл из DHT. Каждая итерация должна тем или иным образом проверять работоспособность приложения.

Кроме передачи сообщений у используемого сокета есть важная побочная функция. Из-за того, что контейнеры модели запускаются последовательно, а протоколы динамической маршрутизации требуют времени для первоначальной настройки до начала работы, на момент запуска узлов моделируемой распределённой системы сеть между ними может быть не готова к работе. Для того, чтобы избежать такой ситуации, при запуске контейнеров, соответствующих узлам приложения, система моделирования с одной стороны дополнительно модифицирует стартовый скрипт таким образом образом, чтобы непосредственная работа узла приложения началась только при наличии в файловой системе IPC-сокета /lab/lab.sock, а с другой — предварительно проверяет его отсутствие до старта модели и создаёт его только после запуска всех контейнеров и проверки корректной работы моделируемой сети.

3.4 Пользовательский интерфейс подсистемы мониторинга

Для демонстрации пользователю данных, полученных от узлов моделируемого приложения, был разработан веб-интерфейс подсистемы мониторинга, который отображает полученные сообщения с помощью трёхмерного графа, построенного на основе логической структуры виртуальной сети, учитывая при этом возможность работы с большим количеством узлов.

В его основе лежит Jgraph, минималистичная Java Script библиотека, предназначенная для визуализации трёхмерных графов в браузере. Она была адаптирована под нужды системы мониторинга, в частности была добавлена поддержка новых форм узлов графа и изменения цвета и размера узла. Форма узла графа зависит от того, соответствует ли контейнер, которой он отображает, узлу моделируемого приложения или роутеру вир-

туальной сети, а цвет и размер — соответственно от значений полей status и traffic, полученных в сообщении от узла. Кроме того, роутеры, на которые были наложены ограничения качества сети, также будут выделены на графе цветом, что позволит сделать отображение текущего качества работы сети более наглядным. Рис. 4 и рис. 5 приводят примеры визуализации моделей с помощью Jgraph.

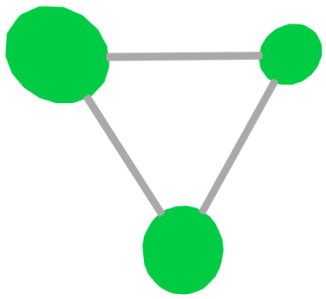

Рис. 4: Отображение простой модели в системе мониторинга. Зелёный цвет узлов сигнализирует о нормальной работе, а размер узла соответствует доле проходящего через него трафика системы.

Кроме трёхмерного графа, пользовательский интерфейс системы мониторинга включает в себя панель просмотра логов, которая состоит из отдельной вкладки для каждого узла, в которую сохраняется текстовое поле log из каждого сообщения, присланного узлом. Для упрощения работы с большим количеством узлов, панель просмотра логов предоставляет пользователям возможность поиска вкладки с логами по имени соответствующего узла. Панель логов изображена на рис.6.

Таким образом, вся полнота информации, полученной от компонентов приложения, находит ту или иную визуализацию в веб-интерфейсе мониторинга модели.

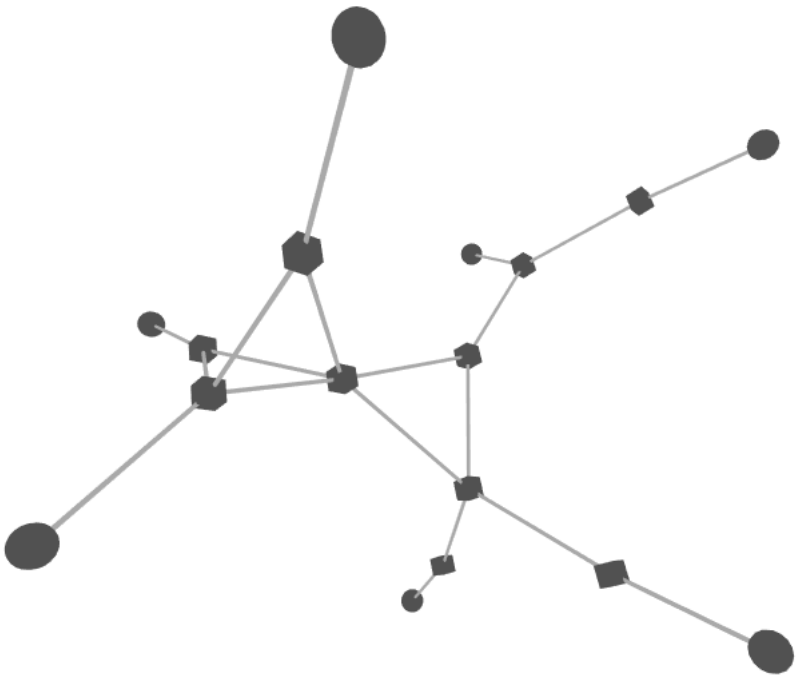

Рис. 5: Визуализация структуры более сложной сети. Круглые узлы графа соответствуют узлам распределённой системы, квадратные - роутерам виртуальной сети.

Рис. 6: Интерфейс панели логов, отображающей работу узла. Из 40 узлов, входящих в эту модель, необходимый найден с помощью филтрации по имени

Глава 4. Демонстрация работы системы

4.1 Подготовка приложения для использования в системе

Для демонстрации созданной системы симуляции было развёрнуто тестовое приложение на основе реализации протокола распределённой хеш таблицы Kademlia на языке Python [22]. Подготовка приложения к тестированию заключалась в создании демонстрационного сценария проверки функционала системы и сборки docker-образа со всем необходимым про-

граммным обеспечением.

Разработанный сценарий тестирования DHT Kademlia с одной стороны запускает сервер Kademlia, который обеспечивает работоспособность сети, а с другой - создаёт клиента и затем в бесконечном цикле проверяет корректность работы системы, последовательно генерируя случайные текстовые данные, сохраняя их в DHT и затем загружая для сравнения. Как можно заметить, каждая такая итерация полностью вмещает в себя диагностику качества работы остальных узлов DHT Kademlia с точки зрения узла, с которого проходит тестирование.

Кроме испытания приложения, сценарий на каждой итерации посылает сообщения в систему мониторинга, в которых ненулевое значение поля status - равно предаёт 0, если приложение работает нормально, и код ошибки в противоположном случае, поле traffic вычисляется как размер хранилища, связанного с ранее запущенным сервисом, а текстовое поле log дублирует поле status, хоть и несколько более развёрнуто.

4.2 Описание моделей, используемых в тестировании

Для тестирования и демонстрации возможностей системы были построены три модели возрастающей сложности. Первая представляет из себя три одинаковых узла, находящихся в одной подсети. Вторая модель демонстрирует возможности системы MADT по организации работы протоколов динамической маршрутизации и включает в себя оверлеи всех трёх типов. Представление логической структуры этой сети в подсистеме мониторинга было показано ранее на рис. ?. Схема распределения оверлеев динамической маршрутизации и автономных систем BGP показана на рис.8

Последняя модель предназначена для демонстрации использования программного интерфейса определения моделей для процедурной генерации сетей большого масштаба. Она была получена следующим образом: для каждого из “внешних” узлов сети сначала были созданы 3 новых “внешних” узла, после чего он был соединён с ними новой подсетью и переведён в перечень “внутренних” узлов. Алгоритм начинает свою работу с единственного

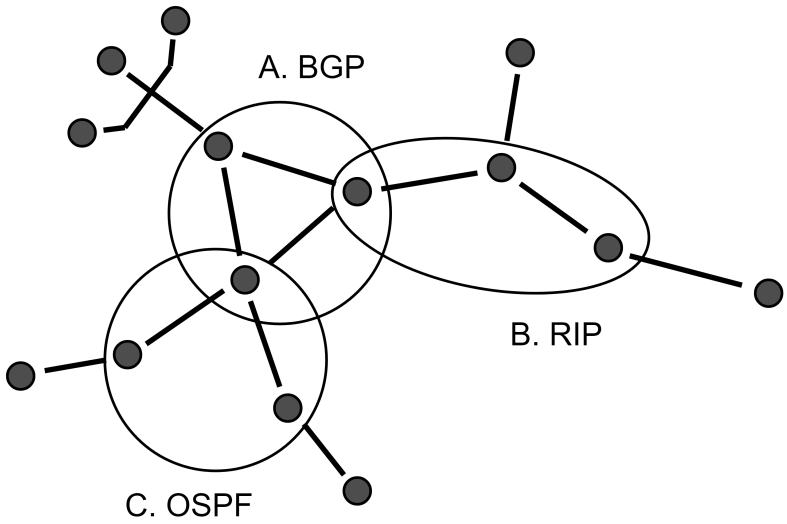

Рис. 7: Схема оверлеев динамической маршрутизации модели

“внешнего” узла, который впоследствии становится центром сети. После N итераций все “внутренние” узлы становятся роутерами и обеспечивают связь между всеми внешними узлами с помощью протокола динамической маршрутизации OSPF, а “внешние” узлы становятся узлами тестируемого приложения.

В тестировании была использована сеть из 27 узлов, полученная после трёх итераций работы алгоритма. Рис.?? демонстрирует представление логической структуры такой сети в интерфейсе подсистемы мониторинга.

4.3 Ход тестирования приложения

Тестирование первой из описанных ранее моделей заключалось в кратковременном отключении одного из трёх узлов от остальных путем установки для него уровня потери пакетов, равного 100%. При этом было обнаружено, что данная реализация протокола Kademlia не осуществляет переподключения к другим узлам в случае потери связи. Сам по себе этот факт не опасен, но несколько уменьшает удобство использования этой библиотеки.

Работа со второй моделью также заключалась в кратковременном отключении отдельных узлов приложения от сети. Тут также была замечена невозможность переподключения узла, как и в прошлом случае, но

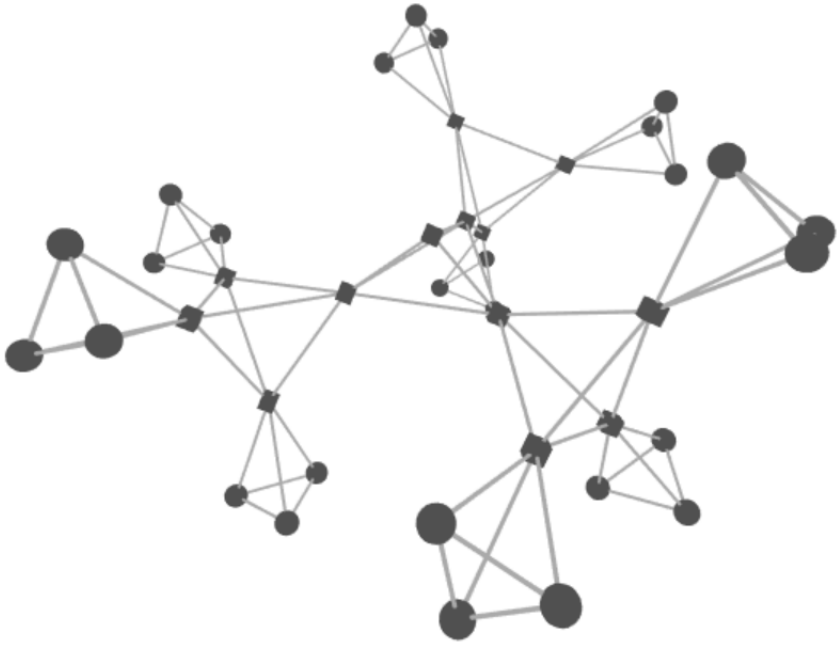

Рис. 8: Процедурно сгенерированая модель, состоящая из 27 узлов и 13 роутеров

возросший масштаб модели позволил обнаружить новый недостаток библиотеки - задержки в работе всех узлов приложения, возникающие после отключения одного из них, как показано на рис. 9.

Тестирование третьей модели было построено иначе. Вместо отключения узлов моделируемой системы, нарушалась работа роутеров, соединяющих разные сегменты сети. Как можно наблодать на рис.10, данном случае Kademlia показала себя в выгодном свете - даже при сегментации нижележащей сети, узлы Kademlia в изолированных участках продолжают работу независимо друг от друга.

4.4 Результаты тестирования

В ходе тестирования DHT Kademlia было обнаружено, что серверная часть данной реализации протокола не может самостоятельно восстановить связь с остальными узлами сети в случае обрыва соединения. Кроме того, отключение одного узла в малых сетях вызывает кратковременные нарушения в работе остальных. С другой стороны, было показано, что данная реализация может работать стабильно в случае потери связи между сег-

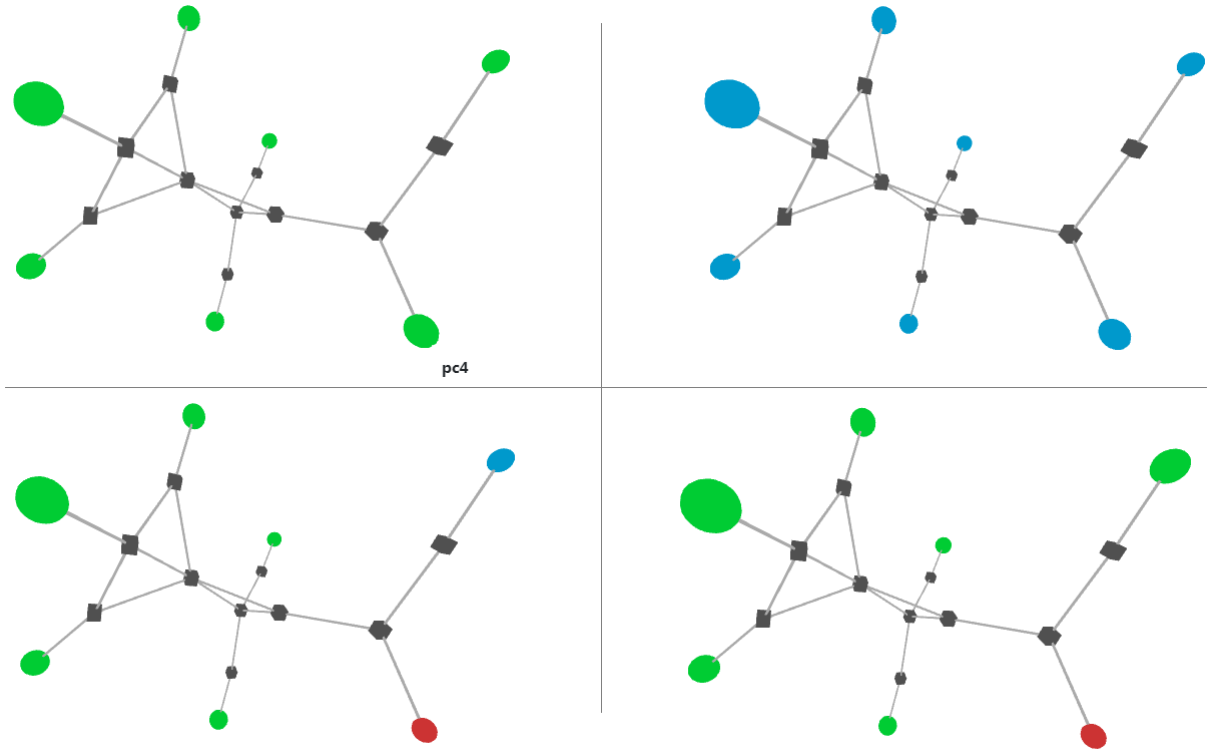

Рис. 9: Визуализация задержек в работе DHT Kademlia, вызванных потерей связи с одним из узлов, отмеченым красным цветом. Последовательно показаны: нормальная работа системы, полная остановка работы на всех узлах, частичное и полное восстановление работы.

ментами сети.

Так как серверная часть данной реализации протокола не пытается восстановить потерянные соединения, а сегментация системы не влияет на её работоспособность с точки зрения конечных пользователей, возможна ситуация, когда нестабильность работы физической сети приведёт к сегментации сети узлов DHT Kademlia, которая отрицательно отразится на её производительности.

Этот пример демонстрирует, как разработанная система симуляции распределённых приложений MADT позволяет обнаружить неожиданное поведение в работе низкоуровневых библиотек, которое может иметь ещё более непредсказуемые последствия в использующих их программных решениях более высокого уровня.

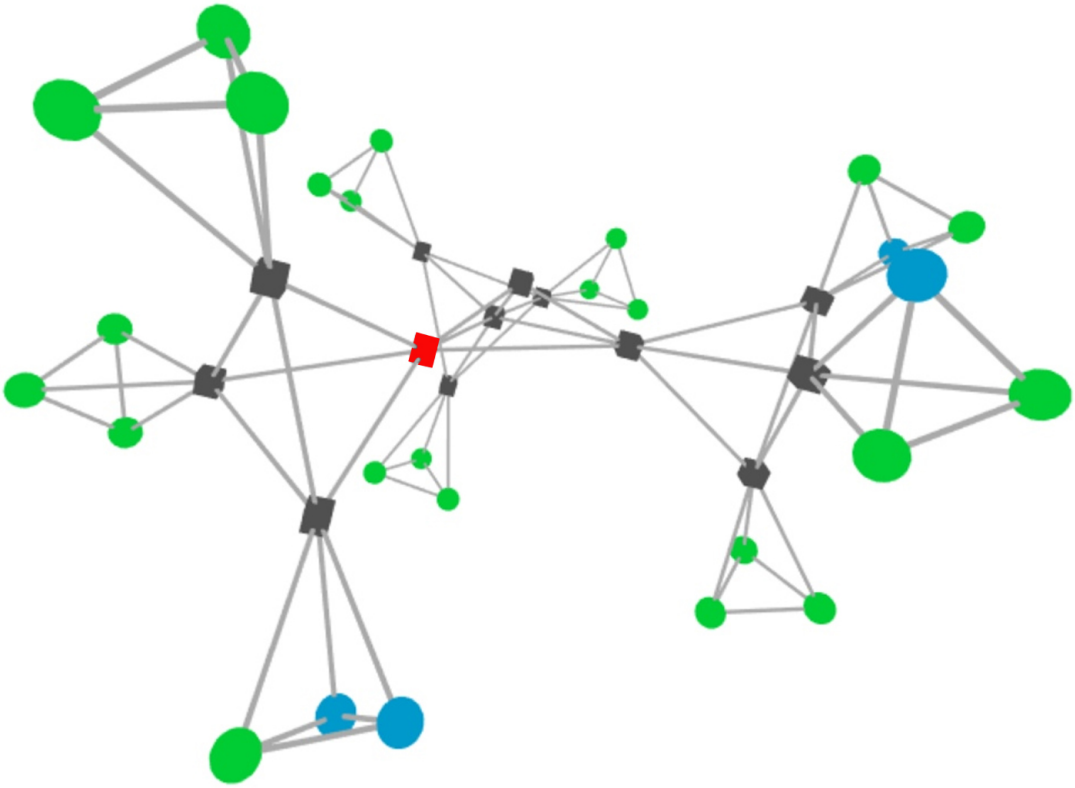

Рис. 10: Работа DHT Kademlia при сегментации сети. Роутер, выведенный из строя отмечен красным цветом

Выводы

Зная архитектуру системы и технологии, на которых она основана, можно сделать предварительные выводы о её возможных сильных и слабых сторонах, которые будут напрямую следовать из особенностей использованных решений.

В частности, применение технологии контейнеризации увеличивает эффективность использования ресурсов относительно виртуальных машин [?] и позволяет гибко управлять качеством работы моделируемых сетей средствами ядра Linux. С другой стороны, оно связано с риском безопасности системы, особенно если речь заходит об открытии доступа к docker-серверу по протоколу TCP [24]. Из-за этого разработанная система непригодна для запуска приложений, полученных из недоверенных источников,

а в случае, когда для моделирования необходимо использование вычислительного кластера, необходима предварительная настройка docker-серверов на его узлах, позволяющая проводить аутентификацию пользователей по SSL-сертификату.

Использование библиотеки ZMQ для передачи сообщений улучшает масштабируемость системы и позволяет использовать сценарии тестирования, написанные более, чем на 40 языках, для которых ZMQ имеет программные интерфейсы. Однако использование сокета в качестве семафора для запуска лаборатории и связанная с этим необходимость туннелирования IPC-сокетов через TCP-соединение могут оказать негативное влияние на скорость передачи сообщений от узлов приложения, моделируемых на узлах кластера.

Организация работы виртуальных приватных сетей, соединяющих контейнеры запущенные на разных узлах кластера с помощью WireGuard VPN, в свою очередь, приводит к трате времени на обязательное сквозное шифрование трафика, которое может увеличить задержки передач сообщений в виртуальных сетях. С другой стороны, реализация WireGuard как модуля ядра убирает необходимость дополнительных требований к docker-образам, используемых в системе.

Если говорить об архитектурных решениях, принятых при разработке системы, то самым важным, является разделение системы на независимые компоненты: API определения модели, API запуска и остановки модели и пользовательский веб-интерфейс управления качеством работы сети и мониторинга работы модели. Это решение позволяет использовать компоненты отдельно друг от друга. Так, модель может быть создана и собрана на рабочем компьютере, а запущена на университетском кластере, эксперименты с моделями небольшой сложности могут также проводиться на рабочем компьютере веб использования веб-интерфейса исключительно в системном терминале. Единственным необходимым исключением является невозможность использования веб-интерфейса отдельно от API запуска и остановки модели.

Передача модели между этими компонентами в собраном виде, состоящем из минимально необходимой для запуска модели информации. Это с

одной стороны позволяет ускорить запуск модели, но с другой не позволяет редактировать её. Это связано с тем, что необходимость изменить модель возникает реже, чем происходит её запуск и остановка, а изменяемое представление модели может быть восстановлено с помощью того же скрипта на языке Python, которым она была изначально сгенерирована.

Кроме того, необходимо отметить, что система реализована таким образом, что единственные требования для docker-образа для его использования в системе, это доступность утилит ip и iptables. Они доступны в пакетном менеджере любого дистрибутива linux, так что требование легко выполнимо.

Заключение

Построенная в ходе данной работы система моделирования распределённых приложений MADT позволяет моделировать работу приложений с большим количеством узлов при разной логической структуре нижележащей сети и изменять качество работы сети в реальном времени. Это даёт разработчикам таких систем испытывать их работу в условиях, приближенным к реальным, в частности при высоких нагрузках и плохой или переменчивой работе сети. Возможность использования динамической маршрутизации и IP-маскарадинга дополнительно увеличивает реалистичность моделей относительно реальных распределённых приложений, развёрнутых поверх сети интернет.

Тестирование с помощью MADT может быть полезным в нескольких случаях. Прежде всего, оно может помочь повысить скорость разработки новых распределённых систем, позволяя заранее находить проблемы при работе приложения с сетью и упрощая проведение стресс-тестирования. Другим возможным применением системы является анализ защищённости распределённых приложений относительно атак сетевого плана, возможный благодаря наличию возможности динамически регулировать качество работы отдельных сегментов сети и поддержке моделирования протоколов динамической маршрутизации. Кроме того, возможность моделирования работы разных технологий поверх сетей произвольной логической структуры делает возможным использование MADT для бенчмаркинга распределённых систем в специфических условиях. Это может значительно упростить пользователям выбор распределённой системы, подходящей для их условий.

Список литературы

[1] Etiemble, Daniel. «45-year CPU Evolution: one law and two equations.»arXiv preprint arXiv:1803.00254 (2018).

[2] 13. Waldrop, M. Mitchell. «More than Moore.»Nature, vol. 530, no. 7589, 2016, p. 144+

[3] K. Skala, D. Davidović, T. Lipić and I. Sović, «G-Phenomena as a Base of Scalable Distributed Computing —G-Phenomena in Moore's Law,»International Journal of Internet and Distributed Systems, Vol. 2 No. 1, 2014, pp. 1-4

[4] Nielsen, «J. Nielsen's law of internet bandwidth.»[Электронный ресурс]: URL: http://www.useit.com/alertbox/980405.html (дата обращения: 24.05.2018).

[5] A. Shurov, D. Malevanniy, O. Iakushkin, and V. Korkhov, «Blockchain network threats: the case of PoW and Ethereum»in Proceedings of International Conference on Computational Science and Applications'19, Lecture Notes in Computer Science, in press

[6] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. [Электронный ресурс]: URL: http://bitcoin.org/bitcoin.pdf (дата обращения: 24.05.2018).

[7] Apostolaki, Maria, Aviv Zohar, and Laurent Vanbever. «Hijacking bitcoin: Routing attacks on cryptocurrencies.»2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017

[8] Global Bitcoin Nodes Distribution - Bitnodes [Электронный ресурс]: URL: http://bitnodes.earn.com (дата обращения: 24.05.2018).

[9] Rekhter, Yakov, et al. «Application of the border gateway protocol in the internet.»(1990).

[10] Henderson, Thomas R., et al. «Network simulations with the ns-3 simulator.»SIGCOMM demonstration 14.14 (2008): 527.

[11] Alberto Montresor and Mark Jelasity. «PeerSim: A scalable P2P simulator»In Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09), pages 99–100. Seattle, WA, September 2009.

[12] M. Feldotto, K. Graffi. «Comparative Evaluation of Peer-to-Peer Systems Using PeerfactSim.KOM »In Proc. of IEEE International Conference on High Performance Computing and Simulation (IEEE HPCS'13), 2013

[13] Dinh, Tien Tuan Anh, et al. «Blockbench: A framework for analyzing private blockchains.»Proceedings of the 2017 ACM International Conference on Management of Data. ACM, 2017.

[14] Lyubomir Stoykov, Kaiwen Zhang, and Hans-Arno Jacobsen. «VIBES: fast blockchain simulations for large-scale peer-to-peer networks: demo»In Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos (Middleware '17). ACM, New York, NY, USA, 19-20.

[15] Docker: Enterprise Application Container Platform [Электронный ресурс]: URL: https://docker.org (дата обращения: 24.05.2018).

[16] Docker SDK for Python: A Python library for the Docker Engine API [Электронный ресурс]: URL: https://docker-py.readthedocs.io/en/stable/ (дата обращения: 24.05.2018).

[17] WireGuard: fast, modern, secure VPN tunnel [Электронный ресурс]: URL: http://wireguard.com/ (дата обращения: 24.05.2018).

[18] Quagga Software Routing Suite [Электронный ресурс]: URL: https://www.nongnu.org/quagga/ (дата обращения: 24.05.2018).

[19] Mayer, C., Mayer, R., Bhowmik, S., Epple, L., Rothermel, K. «HYPE: Massive Hypergraph Partitioning with Neighborhood Expansion »In 2018 IEEE International Conference on Big Data (Big Data) (pp. 458-467). IEEE.

[20] Welcome | Flask (A Python Microframework) [Электронный ресурс]: URL: http://flask.pocoo.org (дата обращения: 24.05.2018).

[21] T. Hombashi «tcconfig: A tc command wrapper. »[Электронный ресурс]: URL: https://tcconfig.readthedocs.io/en/latest/ (дата обращения: 24.05.2018).

[22] Maymounkov, Petar, and David Mazieres. «Kademlia: A peer-to-peer information system based on the xor metric.»International Workshop on Peer-to-Peer Systems. Springer, Berlin, Heidelberg, 2002.

[23] Felter, Wes, et al. «An updated performance comparison of virtual machines and linux containers.»2015 IEEE international symposium on performance analysis of systems and software (ISPASS). IEEE, 2015.

[24] Bui, T. «Analysis of docker security »arXiv preprint arXiv:1501.02967